The Amorphous FPGA Architecture

Mingjie Lin
Department of Electrical Engineering
Stanford University, CA 94305
mingjie@stanford.edu

ABSTRACT

This paper describes the *Amorphous* FPGA, an innovative architecture attempting to optimally allocate logic and routing resource on per-mapping basis. Designed for high performance, routability, and ease-of-use, it supports variable-granularity logic blocks, dedicated wide multiplexers, and variable-length bypassing interconnects with a symmetrical structure. Due to its many unconventional architectural features, the amorphous FPGA requires several major modifications to be made in the standard VPR placement/routing CAD flow, which include a new placement algorithm and a modified delay-based routing procedure. It is shown that, on average, an FPGA with the amorphous architecture can achieve a 1.35 times improvement in logic density, 9% improvement in average net delay, and 4% improvement in the critical-path delay for the largest 20 MCNC benchmark circuits over an island-style baseline.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: [Types and Design Styles]

General Terms

Design, Experimentation, Measurement, Performance

Keywords

FPGA, architecture, amorphous, performance analysis.

1. INTRODUCTION

Despite many advantages of FPGA, the huge performance and cost-efficiency gap between FPGAs and ASICs severely limits its application. Previous studies [1, 2] have shown that without innovations in FPGA architecture, advances in device technology alone can not significantly shrink this gap. Unfortunately, optimizing FPGA architecture proves to be quite challenging mainly because an FPGA's overall performance is jointly determined by many factors including logic block, IO block, clock network, and routing architectures, etc. As a result, the architecture of today's FPGAs, although enhanced with extra features such as block RAMs and embedded microprocessor, still very much resembles the one used in the first generation of FPGAs with similar well-structured island style, in which an array of logic blocks are surrounded by pre-fabricated programmable routing channels as illustrated in Figure 1.

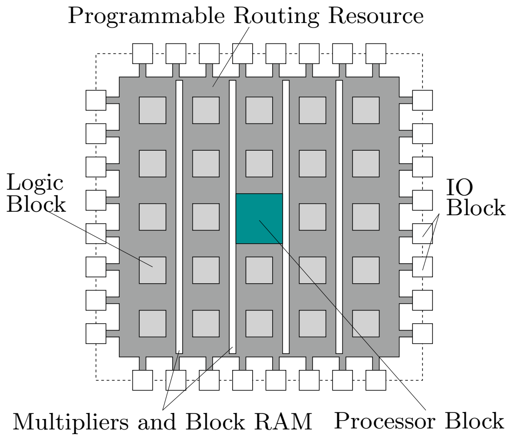

Figure 1: A generic island-style FPGA.

Conventional island-style FPGA poses several challenges to architecture design. The first challenge is how to find a good balance between flexibility and efficiency in terms of area, performance and power, i.e., how to optimally allocate hardware resource between logic and interconnects while still achieving maximum overall performance for a given set of target designs. Conventionally, FPGA has a strict separation between logic and routing resources. This division is determined before the chip fabrication and is fixed at the configuration time. Despite of many advances in device technology over the last decade, a large proportion of the silicon area (≈60-80%) is always devoted to routing resources in order to ensure sufficient routability [3]. Meanwhile, the logic blocks are becoming ever more complex, attempting to perform coarse-grain functions and therefore lighten the stress on the routing resources, but often end up being under-utilized. Given a fixed amount of hardware, how to optimally partition them between logic and routing remains an open problem in FPGA architecture design. This challenge is further complicated by the factor that a generic FPGA family often needs to maximize the application spectrum covering both control- and data-path applications. One conceptually appealing idea is to remove the hard boundary between logic and routing, and therefore permits applications with regular logic structures to more efficiently utilize silicon area, while still permitting the use of many interconnects at the expense of logic for random logic circuits. It should be noted that trading hardware resource between routing and logic is not totally new idea, early Pilkington architecture and Triptych [4] were two early attempts to follow this idea and have shown significant density advantages over traditional island-style. More recently, [1, 5, 6] follow the similar approach of spreading out logic to match routing demand.

The second challenge is to determine the granularity of logic blocks in an FPGA. All prominent FPGAs[7, 8, 9, 10] today have fixed and uniform logic granularity for each logic block. From the architecture point of view, coarse-grain blocks have much less stress on the placement and routing but often result in long internal logic delays and under-utilization for designs in small size, whereas fine-grain logic blocks can achieve shorter internal delay but often requires excessive amount of routing resource in order to successfully route a circuit. From the application point of view, data-path functions, in particular arithmetic functions, often operate on coarser arguments than control-path logic and are usually realized by fine-grain logic elements, while the implementation of control-path logic mostly benefits from coarser granularity. A rather interesting question is whether the logic blocks in an FPGA should be heterogeneous or homogeneous in size. There are two architectural reasons to believe that an FPGA with heterogeneous blocks can potentially provide superior speed and density: (i) Different kinds of logic may be more efficiently implemented with different kinds of blocks. (ii) Previous studies have shown that coarse-grain blocks exhibit superior speed to fine-grain blocks, yet the smaller blocks have better density[11, 12]. A mixture of the two may provide superior speed-area trade-off. Partially motivated by the above observations, Hutton *et al* [13] proposed a new adaptable FPGA logic element based on fracturable 6-LUTs, which fundamentally alters the long-standing belief that 4-LUT is the best choice for area/delay trade-off.

The third challenge of designing an island-style FPGA is to determine the optimal segmentation of routing interconnects. In conventional routing architecture, each routing channel consists of a group of interconnects with variable lengths. For example, Virtex II [8] has 16 Single, 40 Double, 120 HEX, and 24 long interconnects in each routing channel. In general, short segments are advantageous to routability but bad for delay and power performance, while long interconnects achieve better delay performance but may result higher power consumptions. For a given set of benchmark circuits, what is exactly the optimal segmentation for a particular routing architecture remains an open question.

New Approach

We propose the *Amorphous* FPGA architecture to meet above design challenges. Our objective is to develop an architecture that maximizes the application spectrum for both data-path and control-path applications without compromising performance and area efficiency. The main motivation behind the amorphous architecture is to reduce the significant cost paid for routing in standard FPGAs and translate the saving in hardware usage into performance gain. The central idea of the amorphous FPGA is to make several architectural choices dynamically configurable on a per-mapping basis at configuration time. The concept of this architectural "shapelessness" is illustrated in Figure 2. While in the conventional island-style architecture, there is a strong separation between logic and routing resources, and this resource partition is fixed after the chip fabrication, the amorphous FPGA allows the dynamic resource partition at configuration time. In addition, the amorphous architecture can readily perform several system-level functions such as (i) *dynamic resource allocation* between logic and routing, (ii) *variable-granularity logic blocks*, (iii) *dedicated wide multiplexers*, and (iv) *variable-length interconnect overlay* without passing through switching points.

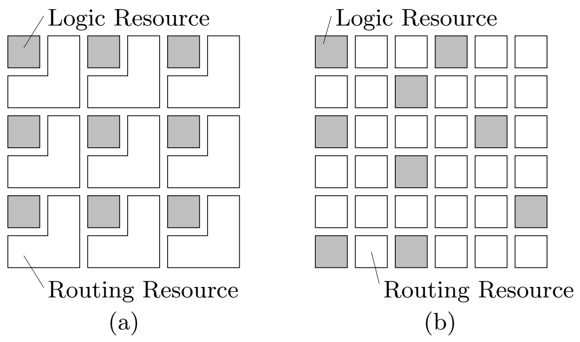

Figure 2: Conceptual picture of FPGA architectures: (a) Conventional island-style FPGA, (b) Amorphous FPGA.

In the following section, we describe the amorphous FPGA architecture in detail. We then illustrate in Section 3 how various system functions can be performed inside an amorphous FPGA. Before presenting the performance comparison results between the amorphous architecture and an island-style baseline in Section 5, we present our placement and routing algorithms in Section 4. Finally in Section 6, we summarize our findings and comment on several open research problems related to the amorphous FPGA architecture.

2. THE AMORPHOUS ARCHITECTURE

As depicted in Figure 3, the top-level architecture of the amorphous FPGA architecture consists of an array of Routing or Logic Element (ROLE) blocks with horizontal and vertical routing channel overlay on the top. Different from the conventional island-style FPGA, the amorphous FPGA replaces logic blocks with specially designed ROLE blocks that allow the dynamic partition of hardware resource between logic and routing on a per-mapping basis after chip-fabrication. Each ROLE block is capable of performing logic only, routing only, or the combination of both tasks.

Routing or Logic Element (ROLE)

As shown in Figure 4, a typical ROLE block contains three types of functional structures: 4-input look-up tables (4-LUTs), flip-flop registers, and MUXes. The main motivation of this design is the observation that the logic capability of a LUT supersedes that of a MUX or a multiple-input switch. As shown in Figure 5, a 4-LUT can readily implement a 2:1-MUX or 4-input Switch. To differentiate from conventional LUTs (look-up tables) and MUXes (multiplexers), we name the structure depicted in Figure 4(b) as MUT (Multiplexer or look-Up Table). Three parameters W, m, and k define the structure of a ROLE block. W denotes the total number of MUXes and MUTs along each side of the ROLE block, m is the number of MUTs on each side, and k is the number of inputs for a MUX or MUT in a ROLE block. Figure 4 depicts a ROLE block with $W = 3$, $m = 1$, and $k = 6$. A ROLE block can be configured into different types of functional blocks. If all 6-MUTs are used as 6-MUXes, then the whole ROLE will behave like a routing block. In contrast, if we use all 6-MUTs as combinations of 4-LUTs and their associated FFs, then the whole ROLE can be looked as a typical logic block with four 4-LUTs. Alternatively, we can partially use 6-MUTs and use the ROLE block as a hybrid

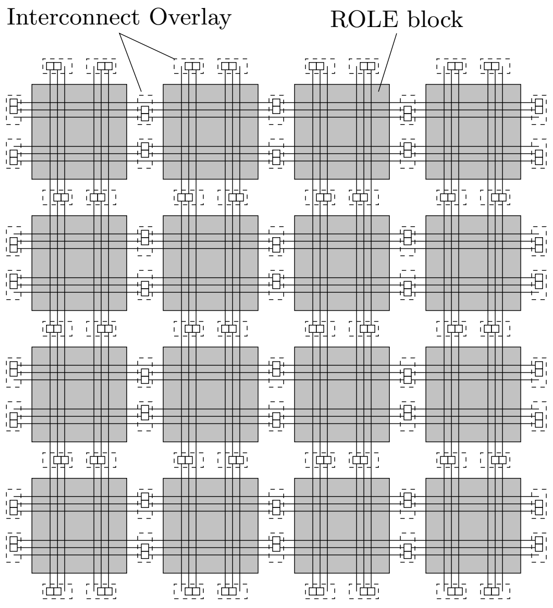

Figure 3: Top-level diagram of an amorphous FPGA architecture.

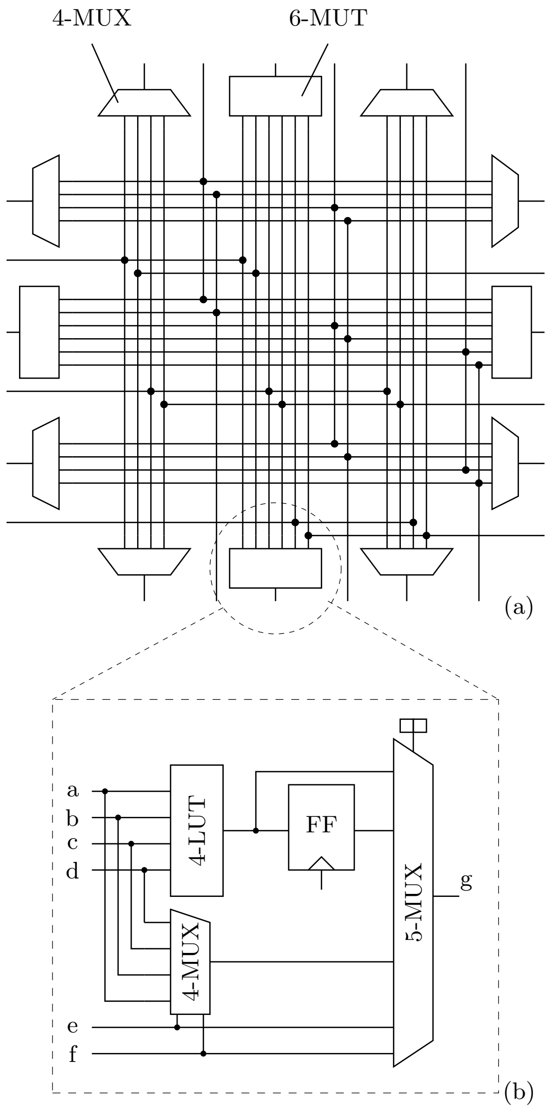

Figure 4: (a) A small example of ROLE block. (b) Structure of a MUT.

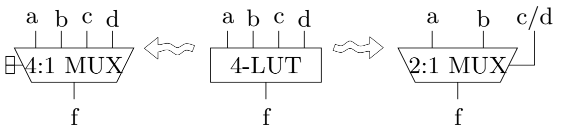

Figure 5: A 4-LUT and a 2:1-MUX or 4:1-Switch it can implement.

of a routing block and a logic block with smaller number of 4-LUTs.

Interconnect Overlay

In modern FPGAs, the routing fabric not only contributes the most to the system delay but also consumes most of the chip area [14, 3]. To make the situation even worse, the fraction of total delay due to routing in an FPGA is increasing with each process generation [15]. Consequently, an FPGA architect must devise routing architectures that are both fast and area-efficient in order to fully exploit the performance and density potential of deep-submicron technologies.

It is known that interconnect segment length can significantly impact the overall performance of an FPGA [16, 17]. Short connections, in general, are advantageous for routing but often result in very densely placed areas where clouds of highly connected logic are placed and cause regional routing problems. Additionally, using short interconnects tends to degrade the overall delay and power performance. On the contrary, long interconnect benefits power and delay but often hurts the routability. Previous studies have shown that optimizing segmentation is quite challenging. We took a different approach and developed a new *bypassing* interconnect for the amorphous FPGA. The key idea is to construct long interconnects without passing through heavy switching points and therefore improve delay and power performance without sacrificing routability. Additionally, the more regular structure in bypassing interconnects makes the delay of long interconnects in the amorphous FPGA much more predictable than that in a conventional segmented routing architecture, and therefore makes it easier for the CAD software to find the most optimal routes. The idea to construct long segments by directly connecting short ones was also explored in [18, 19] under different settings.

As shown in Figure 6, each routing channel comprises only Single and Double segments. Each Single or Double segment consists of two unidirectional wires controlled by two tri-state buffers. Segments can be connected directly to form longer interconnect segments by appropriately setting the states of the tri-state buffers without entering ROLE blocks. This helps reducing the parasitic loading along interconnects due to programming overhead. The segments can also be connected through routing blocks to make bends, fan-out, or connect to logic blocks. The number of interconnect segments in each routing channel is denoted by T and the number of routing tracks in each interconnect segment is r. Figure 3 and Figure 6 show an interconnect overlay of $T = 2$ and a more detailed design of interconnect segment

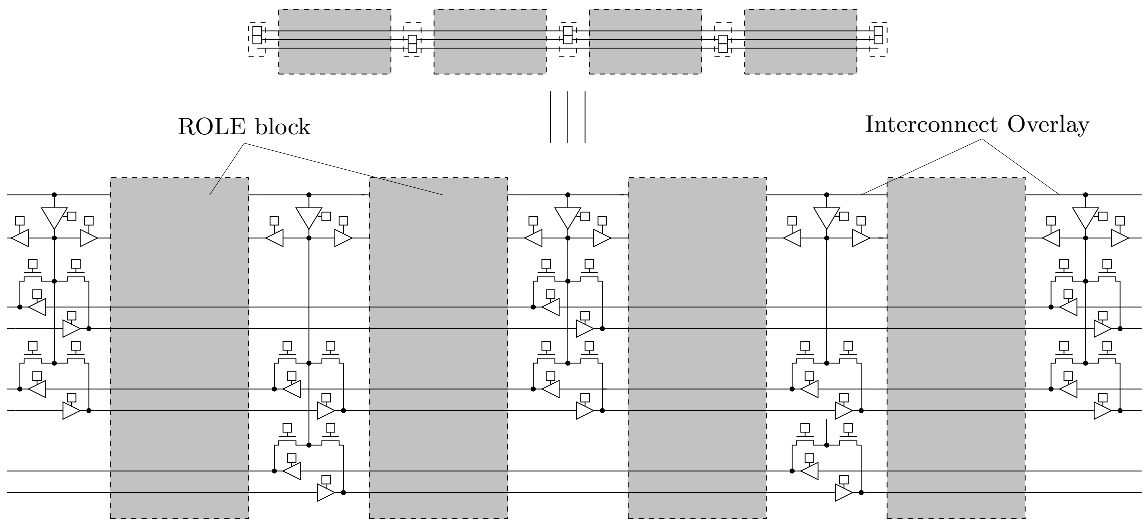

Figure 6: Interconnect overlay and its logic design.

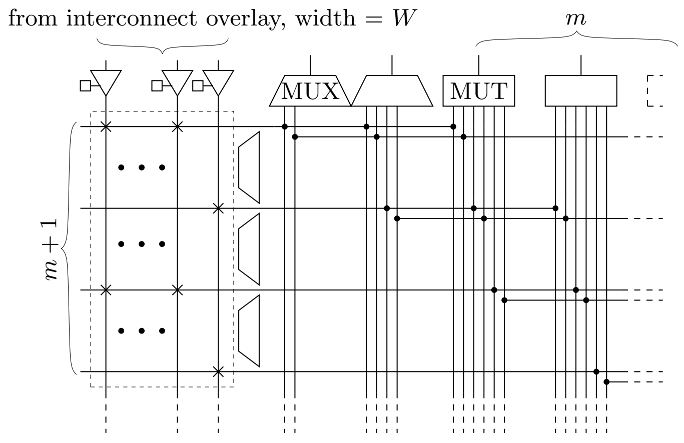

Figure 7: Connections between interconnect overlay and a ROLE block.

of $r = 2$, respectively.

Between Interconnect Overlay and ROLEs

Connections between the interconnect overlay and a ROLE block are made through a structure similar to the conventional Connection Box (CB) as illustrated in Figure 7. For a ROLE block with m MUTs on each side, there are $m+1$ lines crossing over with the W interconnect segments from the overlay. We define F_c as the connecting flexibility of this structure. In Figure 7, $F_c = 0.5$. Note that when the ROLE block is configured as a routing block, this connecting structure enhances the routing capability of the routing block which in the conventional island-style FPGA, if a LB is unused, its associated CB will be idle and only add parasitics to the interconnects.

3. ACHIEVABLE SYSTEM FUNCTIONS

Together with bypassing interconnects, a ROLE block provides additional ability to implement variable-granularity LUTs and wide multiplexers, both of which can potentially reduce the depth of logic circuits and improve delay performance.

Variable-Size LUTs

The number of logic levels significantly impacts delay performance in FPGAs [20]. By collapsing levels of logic, the routed circuits can achieve superior design performance [21]. One effective way to reduce the logic levels in an FPGA is to implement wide-gating functions using variable-input look-up tables. Wider LUT leads to at least two benefits: (i) By directly implementing wider functions in the LUT, the design circuit can reduce the number of logic levels between registers and lead to higher performance. (ii) Because larger LUT reduces the amount of required interconnects, the overall power consumption in the design circuit can be reduced. Most FPGAs before 90's have been based on the same fundamental architecture with 4-LUTs. As a result, functions requiring more than four inputs had to be implemented using a combination of several LUTs and/or multiplexers. As the CMOS device technology scales into deep-submicron domain, many vendors recognize that wider LUT may provide better trade-off between critical path delay and logic density. As a result, the Xilinx Virtex-5 [22] offers 6-input LUTs with fully independent inputs and the Altera Stratix II [13] provides an 8-input fracturable LUT. However, wide-gating functions in [22, 13] come with a price. First, the wide-gating functions are enabled through use of extra dedicated circuitry, which itself adds the hardware cost and can be wasted if not used. Second, although wide-gating function can effectively reduce critical path delay in the design, it is often under-utilized because the implementation of data-path functions, and in particular arithmetic functions, is usually realized by fine-grain elements in practise. In contrast with [22, 13], the amorphous FPGA enables wide-input LUTs without extra circuitry and the MUXes used to construct 6-LUTs from 4-LUTs can be otherwise used for routing if a ROLE is configured as a routing block. The variable-input LUT is a fundamental component of the amorphous FPGA architecture that enables implementing the circuit design with minimal levels of logic without compromising

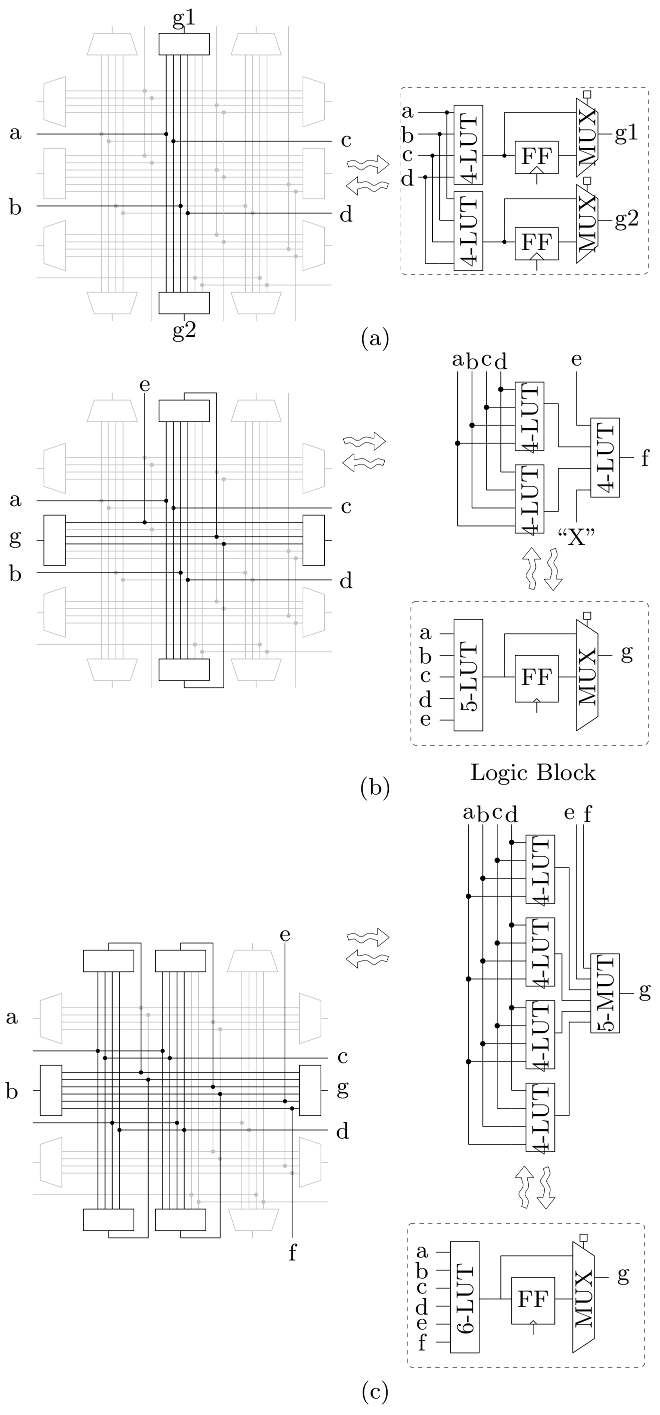

Figure 8: (a) Two different functions with two different outputs for the same four inputs. (b) A 5-input LUT implemented with a ROLE block. (c) A 6-input LUT implemented with a ROLE block.

its logic flexibility.

To further understand the usage of ROLE blocks, we present three examples and compare them with their equivalent implementations with conventional LUT-based logic blocks. In Figure 8(a), the MUT implements two different functions with two different outputs for the same four inputs. As shown in study [22], 5-LUTs can be quite useful in some applications. Conventionally, a 5-LUT can be implemented either (i) with two LUTs and a 2-MUX in a single LUT logic level or (ii) with three LUTs in two logic levels. In Figure 8(b) and (c), we illustrate how a ROLE block implements a 5-input LUT and a 6-LUT, respectively.

Wide Dedicated Multiplexers

Digital circuits often contain many multiplexers and the availability of wide multiplexers can improve the system performance significantly [8]. With conventional 4-LUT, the largest MUX that a single 4-LUT can implement is a 2:1 MUX with the fourth input available as a possible enable. To construct larger MUXes inside an FPGA, the conventional approach is to cascade multiple 4-LUTs. One such example is the Xilinx Virtex architecture [23]. It provides a dedicated 2:1 MUX following every LUT, replacing additional levels of LUT-based logic. One of these LUTs can combine with adjacent LUTs to create a 4:1 mux. As shown in Figure 9(a), a 4:1 MUX can be built by combining the outputs of two 4-LUTs into a third 4-LUT. However, this method adds two full levels of logic delay plus an additional routing delay between the LUTs. A better approach is presented in [24], where Metzgen *et al* showed how to construct a 4:1 MUX using only two 4-LUTs as shown in Figure 9(a). In general, the shortcoming of cascading multiple LUTs is its adverse effect on delay performance and its low area-efficiency. Due to its special design, the amorphous FPGA can easily implement variable-size multiplexers without cascading multiple LUTs and therefore increases MUX speed and density. The reason is because a MUT contains a 4:1 MUX as shown in Figure 4(b). While it may seem that build-in MUX in a MUT can incur extra hardware cost, the keypoint to note is that if this MUX is unused, it can be used as part of routing block and therefore not wasted. To illustrate, we shown in Figure 9(c) the way a wide MUX can be implemented in a ROLE block of the amorphous FPGA.

4. CAD ALGORITHMS

In order to make meaningful FPGA architecture comparison, it is essential that the CAD tools used to place and map circuits into each architecture are of high quality. The routing phase of the amorphous FPGA are largely based on the previously published VPR [25]. Unfortunately, the traditional VPR style CAD flow is not totally suitable for the amorphous architecture because the logic and routing resource has to be partitioned before routing stage. During our research, we also tried several previously published white-space allocation placement algorithms [26, 27] and found the results unsatisfactory for our purpose. As a result, we developed a new algorithm to solve the placement problem for the amorphous FPGA. Given a target design circuit, we start with logic packing assuming homogeneous logic blocks and the same technology packing procedure as in VPR.

Placement Algorithm

Our placement software for the *Amorphous* FPGA is based on a simulated annealing approach with a cost function especially designed to match the routing fabric of the amorphous FPGA. For a island-style FPGA, the placement normally uses metrics such as wirelength as well as some measure of

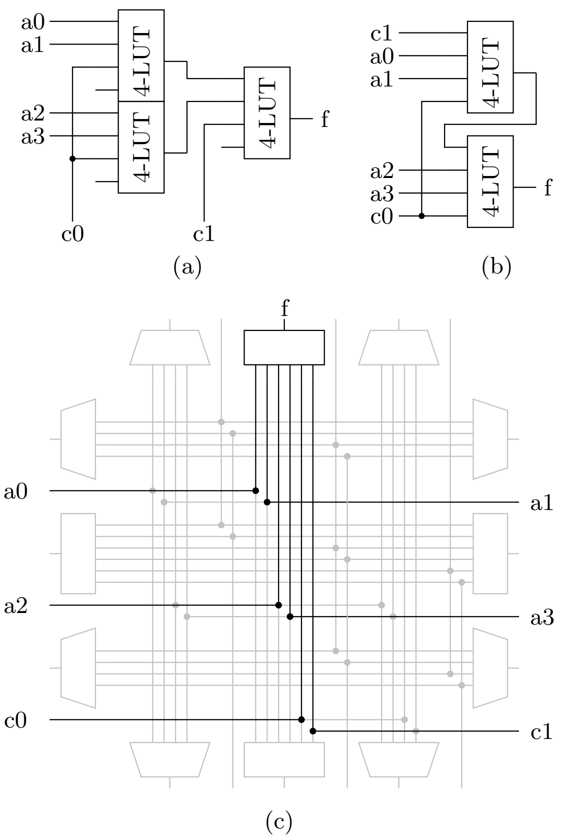

Figure 9: (a) A $4:1$ multiplexer in three 4-LUTs. (b) An efficient $4:1$ multiplexer in two 4-LUTs. (c) A $4:1$ multiplexer implemented with one MUT.

the routability and delay. If applying the same methodology to the amorphous FPGA, minimizing these metrics will cause logic blocks to be placed tightly in the center of the array of ROLE blocks, which almost certainly results in unroutable nets. The placement procedure of the *Amorphous* architecture differs from the island-style FPGA in a fundamental way: its successful routing relies heavily on configuring some ROLE blocks as routing blocks and allocating them to match the routing demand of other RLB blocks configured as logic blocks in the array. To achieve high logic density, the routing resource in an amorphous FPGA should be distributed non-uniformly based on the routing demands between each pair of logic blocks, which is depicted in Figure 2(b). As a result, all ROLE blocks must be configured and positioned as part of the placement process.

To place the ROLE blocks configured as LBs, we first need to quantify the routing resource between two LBs. Figure 10 illustrates two examples, in which ROLE blocks A, B, C, and D in gray color are configured as LBs and all other ROLE blocks in white color are configured as RBs. Quantitatively, the gray area in Figure 10(c) and (d) shows the amount of routing resource between A-D and B-C pairs, respectively. Notice that these gray areas correspond to the Steiner tree cover for these routed nets. Intuitively, a good placement should be such that the size of the gray area between two ROLE blocks is proportional to the signal communication bandwidth between these two ROLE blocks.

We now describe the top-level placement algorithm listed in Algorithm 1. Given a $N \times N$ array of ROLE blocks, we

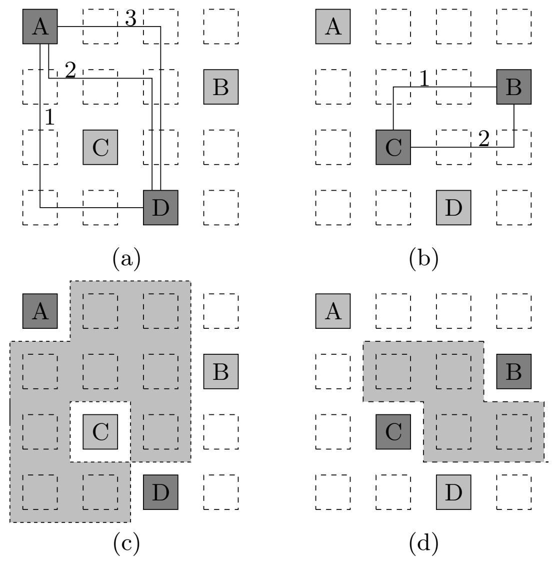

Figure 10: Two examples of availability of routing resource between two LBs.

start with an random placement of ROLE blocks configured as LBs. The corresponding routing graph is then generated and the target design is routed using the standard solution of the Euclidean Steiner tree problem. An initial temperature for simulated annealing is set. We then randomly pick a pair of ROLE blocks and swap them. The designs are then rerouted using Steiner tree algorithm, and the cost metric is re-evaluated. If the metric value is reduced, the swapping is accepted, otherwise it is accepted with a probability that depends on the increase in the value of the cost metric and temperature. The process of ROLE swapping, computing its cost metric, and accepting or rejecting it is repeated until InnerLoopCriterion is false. After exiting the inner loop, temperature is reduced and the process is repeated until the ExitCriterion becomes false.

In the following we describe some of functions referred to in Algorithm 1 in more detail.

ExitCriterion() & InnerLoopCriterion()

ExitCriterion() make sure the freeze_count is less than a predetermined number, which is set equal to 50. InnerLoopCriterion() is true if trials $<$ TRIALS and changes $<$ CHANGES. Both constants TRIALS and CHANGES are related to the problem size. We set TRIALS and CHANGES equal to $10W$ and $0.01W$, respectively.

EvaluateCost()

Given a placement of ROLE blocks and the benchmark design, the first step of evaluating the cost is to perform the routing using the Steiner tree algorithm. It is well-known that most Steiner tree problems are NP-complete, i.e., thought to be computationally hard. To simplify our placement algorithm and to combat the intractability, we use a heuristic[28], in which we compute the Euclidean minimum spanning tree to approximate the Euclidean Steiner tree problem.

Figure 11(a) shows a simple example of a net to be routed and (b) shows the minimum distance Steiner tree that successfully routes this net. Suppose more nets are routed us-

Algorithm 1 The placement algorithm of Amorphous FPGA.

```
1: $p \leftarrow$ RandomPlacement()
2: $T \leftarrow$ InitialTemperature()
3: $g \leftarrow$ g($\mathcal{A}$,$p$)
4: freeze_count $\leftarrow 0$
5: while (ExitCriterion() is FALSE) do
6:    changes $\leftarrow 0$
7:    trials $\leftarrow 0$
8:    $c \leftarrow$ EvaluateCost($g$,$b$)
9:    while (InnerLoopCriterion() is FALSE) do
10:      trials $\leftarrow$ trials + 1
11:      $p_{\text{new}} \leftarrow$ RandomSwap($p$)
12:      IncrementalRoute($g(\mathcal{A}$,$p_{\text{new}})$, $b$)
13:      $\Delta c \leftarrow$ EvaluateCost(g($\mathcal{A}$, $p_{\text{new}}$)) - $c$
14:      if $\Delta c < 0$ /*downhill move*/ then
15:         changes $\leftarrow$ changes + 1
16:         $p \leftarrow p_{\text{new}}$
17:         $g \leftarrow$ g($\mathcal{A}$, $p$)
18:         $c^* \leftarrow$ EvaluateCost(g($\mathcal{A}$, $p_{\text{new}}$))
19:      end if
20:      if $\Delta c > 0$ /*uphill move*/ then
21:         $r \leftarrow$ Random(0,1)
22:         if $r < e^{-\frac{\Delta c}{T}}$ then
23:            $s \leftarrow p_{\text{new}}$
24:            $g \leftarrow$ g($\mathcal{A}$, $p$)
25:         end if
26:      end if
27:   end while
28:   $T \leftarrow$ UpdateTemperature()
29:   if $c^*$ changes then
30:      freeze_count $\leftarrow 0$
31:   end if
32:   if $\frac{changes}{trials} < 0.01$ then
33:      freeze_count $\leftarrow$ freeze_count + 1
34:   end if
35: end while
```

ing Steiner tree algorithm as illustrated in Figure 11(c) [1], we use the following equation to compute the cost of placement. Let $(x_{i,j}, y_{i,j})$ be the cost pair for a ROLE block located at (i, j). $x_{i,j}$ be the total number of routed signals along both x- and y-directions above the ROLE block (i, j), $y_{i,j}$ be the total number of routed signal turns above the ROLE (i, j),

$$\bar{x} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{i,j},\ \bar{y} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} y_{i,j}, \qquad (1)$$

$$c = \sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} x_{i,j}^2 - \bar{x}^2} + \sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} y_{i,j}^2 - \bar{y}^2}. \qquad (2)$$

The value of cost c roughly reflects the variance of the demand for routing resource from all ROLE blocks. Intuitively, small c means high placement quality in terms of the overall routability. There are two main reasons to use routability as the main metric for the cost function in the simulated annealing process: (i) The evaluation of c is more time-efficient in comparison with other delay-related metrics. (ii) As mentioned in Section 2, the bypassing interconnect overlay only contains Single and Double interconnects and therefore the

[1]Only two nets are shown.

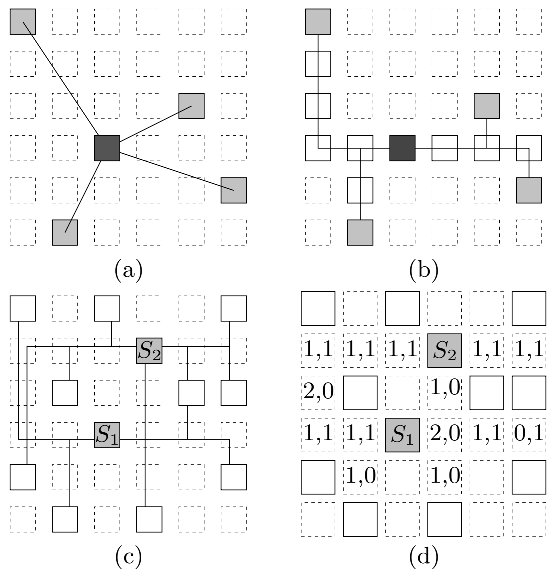

Figure 11: (a) A net to be routed. (b) A routed net using the minimum-distance Steiner tree algorithm. (c) Two routed nets using the minimum-distance Steiner tree algorithm. (d) Computed $((x_{i,j}, y_{i,j}))$ according to Equation 1 and 2 at each block after two nets are routed.

delay is more predicable. It is conceivable that better placement algorithms and cost metrics may exist.

Routing Algorithm

To map designs into the amorphous FPGA, we modified the VPR router [29, 25] to accommodate the differences between the routing architecture of our new amorphous FPGA and the island-based architecture. The routing algorithm initially routes one net at a time using the shortest path it can find without considering interconnect segment or logic block pin overuse. Each iteration of the router consists of sequential net rip-up and re-route according to the lowest cost path available. The cost of using a routing resource is a function of its current overuse and any overuse that occurred in prior routing iterations. By gradually increasing the cost of an oversubscribed routing resource, the algorithm forces nets with alternative routes to avoid using that resource, leaving it to the net that most needs it. The main difference between our router and VPR is that we keep track of visited nodes during the breadth-first-search to improve the run time. More details of this routing algorithm can be found in [30].

5. PERFORMANCE ANALYSIS AND COMPARISON

In this section we compare an amorphous FPGA to the baseline FPGA in terms of routability and delay. We omit the analysis of power consumption for two reasons: the complexity of developing an accurate power estimation model and the limited space for this paper. We assume a 65nm CMOS technology and the Berkeley Predictive Technology Model (BPTM) [31] for devices and interconnects.

As in [30], we assume an island-style FPGA as the baseline architecture, referred to henceforth as *baseline FPGA*, for our performance comparison (see Figure 1). It comprises a 2D array of logic blocks (LBs) interconnected via programmable routing. We assume each LB comprises four logic slices, each consisting of two 4-input Lookup Tables (LUTs), two flip-flops (FFs), and programming overhead. The routing fabric comprises horizontal and vertical routing channels each having sets of Single, Double, HEX-3, and HEX-6 interconnect segments. We classify the interconnects into two groups, *short*, which includes Single and Double FPGA tile width interconnects, and *long*, which includes HEX-3 and HEX-6 interconnects. The segments can be connected to the inputs and outputs of the LBs via *connection boxes* and to each other via *switch boxes*. We assume the MUX-based switch box design described in [32].

We define average net delay for a placed and routed design as the geometric average of all its pin-to-pin net delays, not including LB delay. As in [30], we first use RC models for the interconnect segments and Elmore delay to optimize the connection and switch box device sizes as well as the number and sizes of the buffers for the HEX-3 and HEX-6 segments for a given FPGA array size in each technology node. We then use a modified version of the VPR delay calculation function to compute net delays. All the following performance analysis are performed on 20 MCNC benchmark designs.

(a) The Baseline FPGA

Architectural Parameter	Value
Tile Width (L)	3678λ
LB Buffer Size (b)	7
# Input Pins (K_i)	16
# Output Pins (K_o)	4
SB Density (F_s) [33]	3
Connectivity of CB (F_c) [33]	28
Segment Length Mix	Single (32%) Double (29%) HEX-3 (18%) HEX-6 (21%)

(b) The Amorphous FPGA

Architectural Parameter	Value
Tile Width (L)	2755λ
ROLE Block Width (W)	36
Density of MUT (m)	4
# Inputs of a MUX or MUT	6
Connecting Flexibility(F_c)	0.5
# Interconnect Segments in Each Channel (T)	16
# Routing Tracks per Interconnect Segments (r)	2

Table 1: Baseline 2D-FPGA parameter values.

5.1 Routability and Logic Density

To compare the routability of the new amorphous FPGA to that of the baseline FPGA we placed and routed the 20 largest MCNC benchmark circuits in both architectures. To make a fair comparison, we do not simply compare the minimum number of routing tracks between these two architectures, instead, we compare the minimum required silicon area. For the baseline FPGA, we chose the number of LBs according to the size of benchmark circuit, varied the routing channel width, and found the minimum track count T_min for each design mapped to each architecture and then applied the area model we developed in [30]. In varying the channel width in the baseline FPGA, we maintained the same fractions of each interconnect type (0.32 for Single, 0.29 for Double, 0.18 for HEX-3, and 0.21 for HEX-6).

Because the amorphous architecture is quite different from the conventional island style and the minimum number of routing tracks alone can not accurately indicate routability, we adopted a different approach. While keeping the width of our ROLE block as 36, we find out the minimum number of ROLE blocks that successfully places and routes a benchmark circuit. We then estimate the area for the minimum required amorphous FPGA. For the amorphous FPGA, we assume the architectural parameter values in Table 1 and buffer size 4 for Single interconnects, 6 for Double interconnects, 8 for buffers driving the routing block input, and 6 for shared MUX output buffer. The MUXes and the pass transistor switches that connect segments to routing blocks use size 4 transistors.

Table 2 compares (i) the minimum channel width T_min for the baseline FPGA and the minimum number of ROLE blocks N_min for the new amorphous FPGA, (ii) the geometric average total segment length, $\overline{L}$, used in routing each pin-to-pin net segment, and (iii) the geometric average of the number of bends, $\overline{S}$, used to route each pin-to-pin net. On average, the $\overline{L}$ values in the amorphous FPGA are about 23% longer than the $\overline{L}$ values in the island-style baseline, but because the ROLE block width (2755λ) is about 25% shorter than the tile width of the baseline FPGA (3678λ), the physical signal path are roughly the same.

Table 2: Routability comparison between the amorphous FPGA (NEW) and the baseline FPGA (BL).

Circuit	Baseline			Amorphous		
	T_min	$\overline{L}$	$\overline{S}$	N_min	$\overline{L}$	$\overline{S}$
alu4	55	13.59	3.73	19	15.51	4.07
apex2	59	12.43	3.18	21	14.85	2.98
apex4	57	10.67	2.80	19	12.96	3.15
bigkey	38	20.58	4.68	42	16.08	4.99
clma	79	20.50	4.79	53	35.99	5.09
des	40	18.19	3.30	53	16.25	3.13
diffeq	41	9.22	2.74	18	8.82	3.26
dsip	30	20.06	4.87	38	12.80	5.03
elliptic	78	16.95	3.86	34	27.07	3.91
ex1010	81	14.01	3.54	40	24.60	4.07
ex5p	74	10.46	2.89	35	16.83	3.15
frisc	83	14.61	3.63	36	25.42	4.09
misex3	62	11.77	3.22	18	14.90	2.89
pdc	109	18.70	3.87	43	39.68	3.88
s298	42	13.80	3.47	18	11.35	2.97
s38417	73	10.17	2.58	43	15.04	2.95
s38584	59	12.47	2.87	21	15.49	3.07
seq	71	12.21	3.17	23	18.18	3.02
spla	96	17.82	3.73	40	37.49	4.07
tseng	41	10.62	3.04	18	9.86	3.06

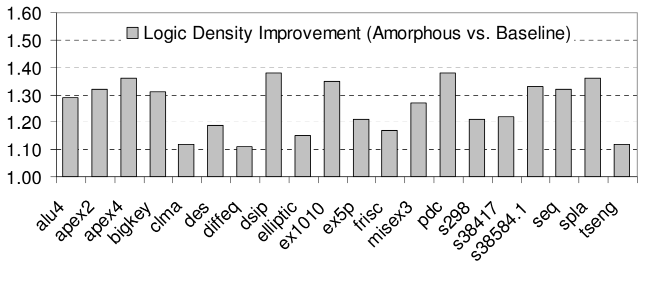

Figure 12: Logic density improvement of the Amorphous FPGA over the baseline FPGA for MCNC benchmark circuits.

In Figure 12, we plot the Logic density improvement of the Amorphous FPGA over the baseline FPGA for 20 MCNC benchmark circuits. By logic density improvement we mean the ratio of the minimum required FPGA area between the baseline and the amorphous FPGA in order to successfully route each benchmark circuit. Note that on average, the new amorphous FPGA requires around 35% less silicon area than the baseline FPGA. The reduction of silicon area is due to several factors. First, the use of shorter segments improves the routability. Second, the capability of the ROLE block to be configured as either logic or routing block and therefore can dynamically adopt to the demand of routing resource between ROLE blocks configured as logic blocks.

Delay Performance

To compare delay performance, for the baseline FPGA, we choose array size of 52×52, which accommodates all the designs with minimum utilization of 12% for the baseline FPGA. We use $F_c = 0.5W$, $F_s = 3$, and $W = 56$ for the channel width. For the segmentation, we assume 18 Single, 16 Double, 10 HEX-3, and 12 HEX-6 segments. The pass-transistor and buffer sizes are listed in Table 1. For the amorphous FPGA, we keep the area the same as the baseline and set the N to be 69. We then use the improvement in the geometric average of the pin-to-pin delays and the critical-path delay as main indicators of delay performance. By improvement here we mean the ratio of the delay in the baseline FPGA to that in the amorphous FPGA. Results for the largest 20 MCNC benchmark circuits are plotted in Figures 13. Note that the improvements over the baseline 2D-FPGA range from 0.89 times to 1.17 times for the geometric average pin-to-pin delay and from 0.87 times to 1.14 times for the critical-path delay. Note for several circuit designs, the delay improvements are actually smaller than 1, which means the delay performance of the amorphous FPGA for these design circuits are actually worse than baseline. On average, there is a 9% delay improvement in pin-to-pin net delay and 4% imrpovement in the critical-path delay for the amorphous over the baseline FPGA.

6. CONCLUSIONS

The central challenge in designing FPGA architecture is how to balance flexibility, performance, and cost. Most FPGA architectures today employ island-style architecture with an array of logic blocks surrounded by pre-allocated routing resources. The *Amorphous* FPGA deviates from this convention and is constructed by an array of ROLE blocks overlayed with an interconnect network. Each ROLE block is capable of either being configured as a logic block or a routing block without extra hardware. As a result, the amorphous FPGA allows dynamic resource partition between logic and routing on per-mapping basis at configuration time, and achieves much better density and delay performance over a baseline island-style FPGA. More interestingly, the amorphous FPGA architecture poses numerous interesting problems open to further investigation:

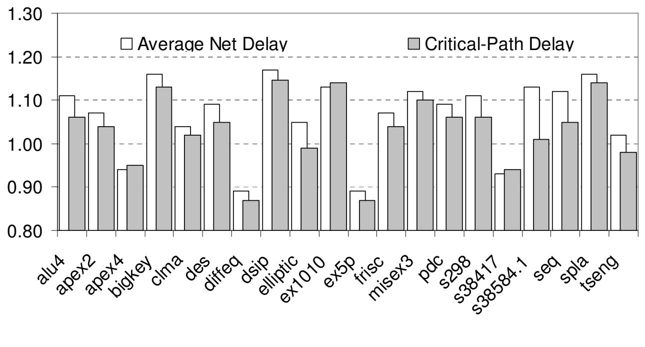

Figure 13: Delay comparison between the Amorphous FPGA and the baseline FPGA for MCNC benchmark circuits.

- Technology packing – Suppose the target design can be packed into logic blocks of variable size, how to determine the optimal size distribution of logic blocks and perform the afterward placement/routing procedure?
- Placement algorithm – Our presented placement algorithm for the amorphous FPGA is a preliminary attempt with only routability being considered. More superior placement and routing algorithms are yet to be developed.
- 3D-implementation – The flexibility of dynamic resource partitioning in the amorphous FPGA largely depends on the amount of memory cells. It is conceivable that a 3D-IC technology with relative cheaper memory cells may prove to be far better implementation technology for the amorphous FPGA.

Acknowledgments

The authors wish to express their thanks to Professor Abbas El Gamal for helpful discussions, Dr. Mike Hutton from Altera for many constructive suggestions, and Dr. Wenyi Feng from Actel for the addition of several key references.

7. REFERENCES

[1] A. Dehon, “Nanowire-based programmable architectures,” *J. Emerg. Technol. Comput. Syst.*, vol. 1, no. 2, pp. 109–162, 2005.

[2] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” in *Proceedings of the 2006 ACM/SIGDA Tenth International Symposium on FPGA*, pp. 21 – 30, 2006.

[3] A. DeHon, “Balancing interconnect and computation in a reconfigurable computing array,” in *Proceedings of the ACM/SIGDA 7th international symposium on Field programmable gate arrays*, 1999.

[4] G. Borriello, C. Ebeling, S. A. Hauck, and S. Burns, “The triptych FPGA architecture,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 4, 1995.

[5] M. Tom and G. Lemieux, “Logic block clustering of large designs for channel-width constrained fpgas,” in

DAC '05: Proceedings of the 42nd annual conference on Design automation, pp. 726–731, ACM, 2005.

[6] D. B. Strukov and K. K. Likharev, "A reconfigurable architecture for hybrid CMOS/Nanodevice circuits," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on FPGA*, pp. 131–140, 2006.

[7] Actel, Inc., "Automotive ProASIC3 flash family FPGAs datasheet," March 2007.

[8] Xilinx, "Virtex-II Pro / Virtex-II Pro X complete data sheet (all four modules)," March 2007.

[9] D. Lewis, E. Ahmed, G. Baeckler, V. Betz, M. Bourgeault, D. Cashman, D. Galloway, M. Hutton, C. Lane, A. Lee, P. Leventis, S. Marquardt, C. McClintock, K. Padalia, B. Pedersen, G. Powell, B. Ratchev, S. Reddy, J. Schleicher, K. Stevens, R. Yuan, R. Cliff, and J. Rose, "The stratix II logic and routing architecture," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, pp. 14 – 20, 2005.

[10] E. Ahmed and J. Rose, "The effect of LUT and cluster size on deep-submicron FPGA performance and density," in *the 2000 International Symposium on FPGA*, Feb. 2000.

[11] D. Hill and N.-S. Woo, "The benefits of flexibility in lookup table-based FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 12, pp. 349–353, Feb. 1993.

[12] J. He and J. Rose, "Advantages of heterogeneous logic block architectures for FPGAs," in *Proc. IEEE Custom Integr. Circuits Conf.*, pp. 741–745, 1993.

[13] M. Hutton, J. Schleicher, D. M. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, M. Bourgeault, A. Lee, H. Kim, and R. Saini, "Improving FPGA performance and area using an adaptive logic module.," in *FPL*, pp. 135–144, 2004.

[14] V. Betz and J. Rose, "FPGA routing architecture: segmentation and buffering to optimize speed and density," in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on FPGA*, pp. 59 – 68, 1999.

[15] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3D-FPGA," in *Proceedings of the 2006 International Symposium on FPGA*, pp. 113 – 122, 2006.

[16] L. Ciccarelli, D. Loparco, M. Innocenti, A. Lodi, C. Mucci, and P. Rolandi, "A low-power routing architecture optimized for deep sub-micron FPGAs," in *Conference 2006, IEEE Custom Integrated Circuits*, pp. 309–312, 10-13 Sept. 2006.

[17] M. Pedram, B. Nobandegani, and B. Preas, "Design and analysis of segmented routing channels for row-based FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, pp. 1470–1479, Dec. 1994.

[18] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Proceedings of the 5th IEEE Symposium on FCCM*, p. 12, 1997.

[19] N. Weaver, J. Hauser, and J. Wawrzynek, "The SFRA: a corner-turn FPGA architecture," in *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pp. 3–12, 2004.

[20] O. Agrawal, H. Chang, B. Sharpe-Geisler, N. Schmitz, B. Nguyen, J. Wong, G. Tran, F. Fontana, and B. Harding, "An innovative, segmented high performance FPGA family with variable-grain-architecture and wide-gating functions," in *Proceedings of the 1999 international symposium on FPGA*, pp. 17–26, 1999.

[21] Xilinx, "The 40% performance advantage of Virtex-II Pro FPGAs over Competitive PLDs." White paper by Xilinx Inc., 2006.

[22] Xilinx, "Achieve higher system performance with the Virtex-5 Family of FPGAs." White paper by Xilinx Inc., 2006.

[23] V. M. K. Kamal Chaudhary, Philip D. Costello, "Programmable circuit optionally configurable as a lookup table or a wide multiplexer," July 2006.

[24] P. Metzgen and D. Nancekievill, "Multiplexer restructuring for FPGA implementation cost reduction," in *Design Automation Conference, 2005. Proceedings. 42nd*, pp. 421–426, 13-17 June 2005.

[25] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, pp. 213 – 222, 1997.

[26] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," in *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, (Washington, DC, USA), pp. 394–401, IEEE Computer Society, 2004.

[27] B.-K. C. X. Yang and M. Sarrafzadeh, "Routability-driven white space allocation for fixed-die standard-cell placement," *IEEE Trans. on CAD*, vol. 22, pp. 410–419, April 2003.

[28] A. Kahng and G. Robins, "A new class of iterative steiner tree heuristics with good performance," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, pp. 893–902, July 1992.

[29] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and routing tools for the Triptych FPGA," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 3, no. 4, pp. 473–482, 1995.

[30] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-D FPGA," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, pp. 216–229, Feb. 2007.

[31] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu, "New paradigm of predictive mosfet and interconnect modeling for early circuit simulation," in *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, pp. 201–204, 2000.

[32] G. Lemieux and D. Lewis, "Circuit design of routing switches," in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-Programmable Gate Arrays*, pp. 19 – 28, 2002.

[33] V. Betz, J. Rose, and A. Marquardt, eds., *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.